# Episode 2.9 – Introduction to Gray Code

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java.

Maybe you don't need to use a number to hold a value. Maybe you just need it to keep tabs on something like a position or a sequence. If you're familiar with programming, you've probably used a for-loop. It's that programming construct that allows us to execute a block of code a specific number of times using a counter to keep track of which iteration is currently being executed. For many applications of the for-loop, the value that is in the counter is of no consequence. It's just being used to keep track of how many times we've passed through the loop.

Counting using integers stored in a processor's memory is a reliable operation. If, however, we are counting in binary using a mechanical device, we could encounter critical errors. Let's start by taking a look at a device called an absolute digital rotary encoder. It's a mouthful of a name, but the device it describes is straightforward. This type of encoder identifies the current angular position of a rotating shaft by converting the position to an n-bit binary number that can be read by a processor. By using n bits to digitally represent the position, it's possible to resolve the angle of the shaft to an arc size of 360 degrees divided by $2^n$. If we need better resolution, we'll need to find an encoder with more bits. If the rotary encoder does not have stops to limit the rotation, then it will pass circularly through the list of integers. After a full rotation, the binary integer output will go from all ones to all zeros or vice versa depending on the direction of rotation.

Now let's get to the problem. In the mechanical world, it is unlikely that all bits of the encoder will change at the exact same instant. If we use the conventional sequence of unsigned binary values to count through the positions, half of the pairs of consecutive integers in this circular list differ by two or more bits. For example, 4 comes after 3, which in binary means 011 is followed by 100. If the processor happens to read the digital output from the encoder at the moment the encoder is changing from 3 to 4, there is a chance that some of the bits may be read at their new levels while the changes in other bits have not yet registered. It's possible, therefore, that the first bit to change could be the most significant bit, and what should have been a 4 is read as a 7. Imagine what could happen if this encoder is being used to identify the position of an accelerator pedal connected to an automotive engine control unit.
The use of Gray code, sometimes referred to as reflected binary code or RBC, has been around for some time, but its name comes from the Bell Labs researcher, Frank Gray, who patented the concept in 1953.[1] The application for which Gray used these codes was to decrease error rates in analog to digital conversion, another of the many applications of Gray code, some of which we will cover in our next episode.
The goal of Gray code is to rearrange all of the unsigned n-bit integers into a circular list so that each element of the list is distinct and that neighboring values differ by exactly a single binary digit. If n=1, the job is easy. Only one bit can change, so the cyclic list is 0 then 1 then 0 then 1 then 0, and so on. This satisfies all of the requirements. This list is not very useful, though, as it only resolves our rotation into

one of the two halves of a 360-degree rotation and contains no directional information. So, let's increase our resolution by adding a second bit.

The cyclic integer sequence 00, 01, 10, 11, 00, 01, and so on has two steps out of the four where both bits change: when going from 01 to 10 and when going from 11 to 00. Fixing this is simple. If we swap the order of 10 and 11, we get the sequence 00, 01, 11, 10, 00, 01, and so on. This takes care of both of the problem steps.

It turns out that this rearrangement of integers allows us to create a Gray code sequence for any number of bits. Each sequence of n bits can be assembled using the previous sequence of n-1 bits. To see how, let's use the two-bit sequence to come up with the three bit sequence. Place a zero at the beginning of each of the patterns of the 2-bit Gray code sequence. This gives us 000, 001, 011, 010. This is the first half of the three-bit sequence. For the last half of the three-bit sequence, begin by reversing the order of the 2-bit sequence, then place a one at the beginning of each of these values. This gives us 110, 111, 101, and 100. Put the two sequences together, and you have a full three-bit Gray code sequence: 000, 001, 011, 010, 110, 111, 101, and 100. Note that this also satisfies the cyclic requirement of the sequence where 100 and 000 differ by only one bit. The operation of reflecting the (n-1)-bit sequence in order to generate the n bit sequence is what gives Gray code its alternate name of reflected binary code.

Things start getting a little out of control when we try to develop Gray code sequences with greater numbers of bits. In our next episode, we will cover some of the conversion methods used to convert back and forth between a Gray code pattern and its corresponding unsigned binary position identifier. We will also touch on more of the applications of Gray code. It goes well beyond just mechanical encoding.

Speaking of mechanical encoders, there's an additional benefit to be had with the use of Gray code. If the digital sensors of the encoder utilize mechanical contact for the on and off settings of each bit, then the reduction in the number of bit flips minimizes the amount of mechanical movements required to encode the position. This should decrease wear and tear on the device, which should increase its operational lifespan.

That brings us to the end of another episode of the Geek Author series on Computer Organization and Design Fundamentals. For transcripts, links, or other podcast notes, please check us out at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until then remember that while the scope of what makes a computer is immense, it's all just ones and zeros.

**References:**

1 – Gray, Frank (1953-03-17). "Pulse code communication" (PDF). U.S. patent no. 2,632,058.